BERT vs CrossEncoder vs ColBERT

BERT: Bidirectional Encoder Representation from Transformer
ColBERT: Contextualized Late Interaction BERT
- `Late` compared to CrossEncoder as in ColBERT document embeddings are precomputed while Cross Encoder calculates similarities via BERT internally

BERT

Nothing changes from Transformer but it's called **'Bidirectional'** because it reads all previous and future tokens. As we see in Transformer, a token embeddings is calculated between this token and entire other tokens.

The term Bidirectional can be better understood in the comparison with GPT (unidirectional) or RNN.

Retrieval with BERT

Document Embeddings are pre-calculated using a BERT model and then indexed into a **Vector DB**. When a query comes in an embedding for the query is calculated using the same BERT model. Similarity between a query and documents are calculated an algorithm the vector DB uses such as ANN, which means we cannot fully trust the score itself while there's a high likelihood of a relevant entity being included.

This is same for ColBERT, but not for Cross Encoder

ColBERT

BERTs are used to save one dense representation per document. ColBERT saves **token-level** embeddings while having smaller dimension. For example, one 1024-dimensional embedding for entire document in BERT, while **(num_token x 128-dimensional)** embeddings in ColBERT.

While we index documents in traditional vector DB, token embeddings will be calculated and indexed when using ColBERT. ANN will be done for this. So not simply work with traditional vector DB because even though it's possible to replace document embedding to token embedding (vector DB does not take care what we save as long as it's an embedding), there wouldn't be a native way to aggregate retrieved token level scores back to a document it consists of.

Calculation

$$\text{score}(Q, D) = \sum_{i=1}^{|Q|} \max_{j \in [1, |D|]} \text{sim}(\mathbf{q}_i, \mathbf{d}_j)$$

q_i is token embedding of a query
d_j is token embedding of a document

Are you worried about just taking one maximum similarity among all the document token embeddings? It used to work even though similarity calculation mechanism is simple but embeddings are contextualized, reflecting complex relationships between tokens.

CrossEncoder

There's no embedding we can see, internally it's calculated though. We will get a single similarity score between a query and a document.

This seems weird? I will make sense once you see the input

```
INPUT of CrossEncoders: [CLS] query tokens [SEP] document tokens [SEP]
```

Calculation

A CrossEncoder takes query + document as input, goes through Transformer, and finally take compressed representation([CLS]) to return a similarity

$$\text{score}(Q, D) = \mathbf{w}^\top h_{[\text{CLS}]} + b$$

Summary

You can naturally think in the context of **RAG** that BERT and ColBERT pre-calculate document embeddings, while a calculation happens only once when CrossEncoder gets a query being impossible to pre-calculate something.

Between BERT and ColBERT, BERT uses document-level embeddings and ColBERT uses token-level embeddings so the former would be more efficient at a cost of retrieval performance.

Due to theirs strength in pre-computation, BERT and ColBERT are used for retrieval. CrossEncoders are used for reranking. ***Aren't you curious why CrossEncoder is used for retrieval if ColBERT is fine-grained comparison at token level?*** The key would be in **INPUT**. In ColBERT (and also in BERT), Query and Documents are embedded independently to each other. So when a model converts a series of tokens into vectors, it cannot see in what context this is used for. CrossEncoder sees all the information from query and a document, which is called joint contextualization. That's why CrossEncoder tends to fit better for reranking.

Aspect	BERT	ColBERT	CrossEncoder
Type	Bidirectional Encoder Representation	Late Interaction Bi-Encoder	Full Cross-Attention Encoder
Architecture Use	Pretrained transformer used as base for various NLP tasks	Uses BERT to encode queries and documents independently, then interacts at token level	Uses BERT to jointly encode query and document together
Embedding Strategy	Single vector per text (e.g., [CLS])	Multiple contextualized token embeddings	Joint representation for query–document pair
Computation Cost	Low (single pass per text)	Moderate (independent encoding + late matching)	High (requires re-encoding every pair)
Use Case	Feature extraction, sentence embeddings, text classification	Efficient retrieval with token-level relevance	Reranking or fine-grained relevance scoring
Storage Requirement	Minimal (single vector per text)	High (stores all token embeddings for documents)	Minimal (no pre-stored embeddings; recomputed per query)

Aspect	BERT	ColBERT	CrossEncoder
Retrieval Efficiency	High	High for large-scale retrieval	Low (computationally expensive for many candidates)
Relevance Accuracy	Moderate	High (better token-level matching)	Highest (full attention across query and document)
Example Application	Sentence transformers (SBERT)	Dense retrieval in information retrieval systems	Reranker in search pipelines (e.g., ColBERT → CrossEncoder)